**ACCEPTING SENTENCE FROM THE USER AND SENDING TO SERVER**

S CODE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <ctype.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

// Function to check for duplicate words and remove them

void remove\_duplicates(char \*message) {

char words[BUFFER\_SIZE][BUFFER\_SIZE];

char unique\_words[BUFFER\_SIZE][BUFFER\_SIZE];

int word\_count = 0, unique\_count = 0, i, j;

char \*token = strtok(message, " ");

// Split the message into words

while (token != NULL) {

strcpy(words[word\_count], token);

word\_count++;

token = strtok(NULL, " ");

}

// Check for duplicates and retain unique words

for (i = 0; i < word\_count; i++) {

int found = 0;

for (j = 0; j < unique\_count; j++) {

if (strcmp(words[i], unique\_words[j]) == 0) {

found = 1;

break;

}

}

if (!found) {

strcpy(unique\_words[unique\_count], words[i]);

unique\_count++;

}

}

// Rebuild the message with unique words

strcpy(message, unique\_words[0]);

for (i = 1; i < unique\_count; i++) {

strcat(message, " ");

strcat(message, unique\_words[i]);

}

}

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Bind the socket to the network

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

while (1) {

// Clear the buffer

memset(buffer, 0, BUFFER\_SIZE);

// Read the message from the client

int valread = read(new\_socket, buffer, BUFFER\_SIZE);

if (valread > 0) {

buffer[valread] = '\0'; // Null-terminate the string

// If the message is "Stop", terminate the server

if (strcmp(buffer, "Stop") == 0) {

printf("Client requested termination. Shutting down...\n");

break;

}

// Process the message to remove duplicates

remove\_duplicates(buffer);

printf("Processed message: %s\n", buffer);

// Send the processed message back to the client

send(new\_socket, buffer, strlen(buffer), 0);

}

}

// Close the connection

close(new\_socket);

close(server\_fd);

return 0;

}

C CODE

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char message[BUFFER\_SIZE];

char buffer[BUFFER\_SIZE] = {0};

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 address from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

printf("\nInvalid address/ Address not supported \n");

return -1;

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

printf("\nConnection Failed \n");

return -1;

}

while (1) {

// Input message

printf("Enter a message (type 'Stop' to terminate): ");

fgets(message, BUFFER\_SIZE, stdin);

message[strcspn(message, "\n")] = 0; // Remove trailing newline

// Send the message to the server

send(sock, message, strlen(message), 0);

// Break if the user enters "Stop"

if (strcmp(message, "Stop") == 0) {

printf("Terminating connection...\n");

break;

}

// Receive the processed message from the server

read(sock, buffer, BUFFER\_SIZE);

printf("Processed message from server: %s\n", buffer);

}

// Close the socket

close(sock);

return 0;

}